Challenge SSTIC 2026 - Step 0 : linenoise

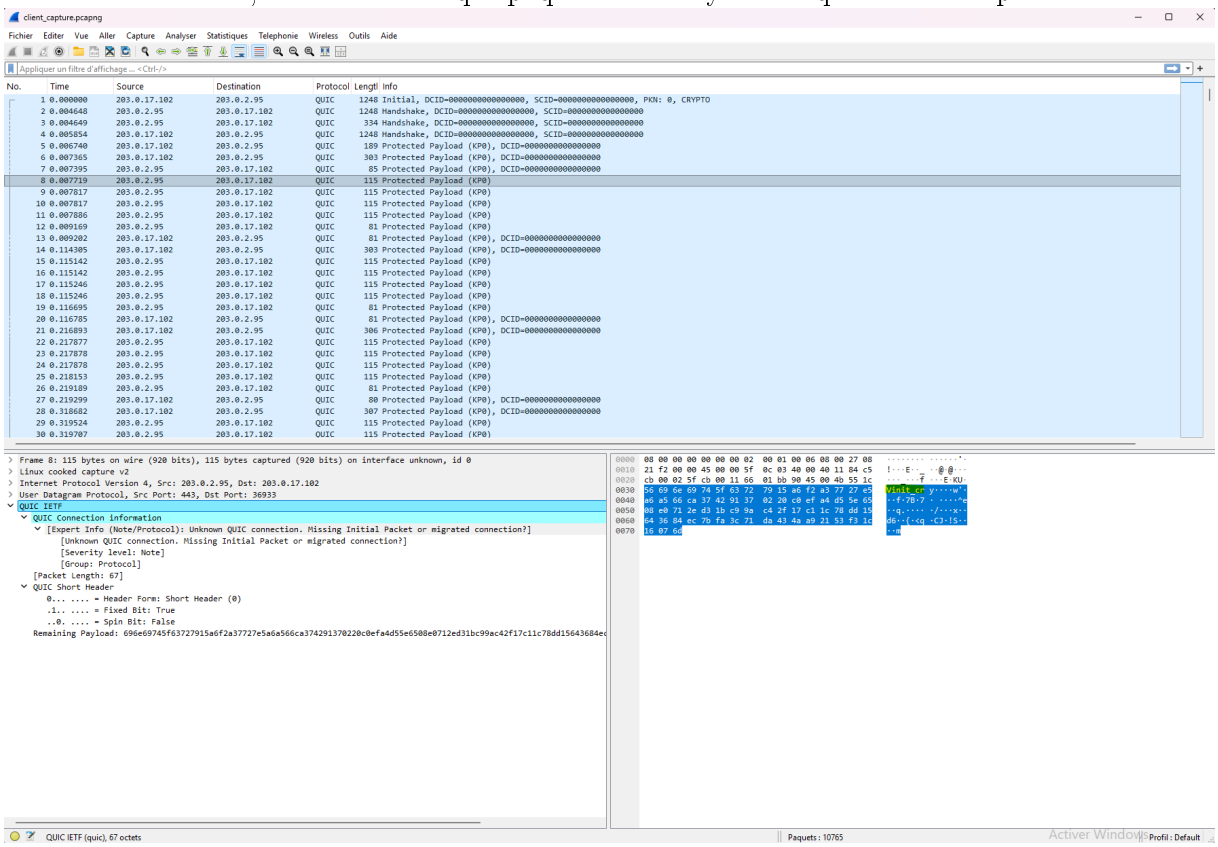
nebucca

```
Hello analyst,

Following an alert from the ABSSI (*Agence Bretonne de la Sécurité des Systèmes d'Information*), we are suspecting a
compromise of one of our contractors. An extract, containing a dubious network traffic, has been supplied. Could you
please have a look at it? As usual, we are looking for an analysis of the exchanges, and any IOCs if relevant.

Please be careful with contained data, if any. The, maybe, compromised contractor is working on a highly sensitive
infrastructure, all information related to this system MUST BE reported at the earliest opportunity.

Thanks for your assistance and your discretion,

-Incident Dispatcher - Investigation des Moyens et Plateformes Sous-traitées
```

1 Analyse

Direction *WireShark* pour l'analyse des traces réseaux. Il s'agit de communication entre deux machines en utilisant le protocol QUIC. Les données sont chiffrées.

Mais chose curieuse, le début de chaque paquet semble systématiquement "compréhensible" :



Du coup l'idée est d'extraire ces parties des différents échanges. Pour ce faire, on va adapter un script [1] afin de ne garder que la partie qui nous intéresse :

```
        local strfld = tostring(ftvbr:string(ENC_UTF_8))
        local strxt = string.sub(strfld, 2, 9)
    tree:add(exfield_string, strxt)
```

Il n'y a plus qu'à lancer le script pour extraire de ce qui nous intéresse :

```
.\tshark.exe -r .\client_capture.pcapng -X lua_script:extract.lua -X
   lua_script1:udp.payload -T fields -e extract.string > extract_full.
   txt
```

On se rend compte assez vite que de nombreuses lignes sont dupliquées (des rejeux ?).

Petit travail de nettoyage et de mise en forme de ce qui s'avère être un malware en python.

On trouve là, le premier flag.

1. https ://wiki.wireshark.org/Lua/Examples#extract-field-values

Challenge SSTIC 2026 - Step 1 : vibe malwaring

nebucca

A cette étape on va se concentrer sur ce qu'on a pu récupérer à l'étape précédente : les sources du malware.
Après un nouveau travail de mise en forme il apparait que le module *config* est chiffré lors du transport.

1 Déchiffrement du module *config*

Comment le client le déchiffre t il ?

```
mod_code_bytes = b64decode(mod_code)
self.crypto.decrypt(self.session_key, mod_code_bytes).decode()
```

D'où vient cette clef de session ?

```
self.session_key = self.crypto.compute_session_key()
```

```
def compute_session_key(self) -> bytes:
    rand = Random(int(time.time()))
    key = rand.randbytes(n=32)
    return key
```

Autrement dit, la clef de session est calculée à partir de la date courante. Il est ainsi possible de retrouver la clef de session à partir de sa date de création. Grâce aux traces réseaux nous avons un horodatage précis de l'événement *set_session_key*.

Avec un script python, petit brute force sur la date à partir de "1771542017" qui nous permet de trouver la clef et de déchiffrer le module de configuration.

2 Domain Generation Algorithm

Dans la configuration on trouve en particulier tout ce qui sert à initialiser la partie DGA (Domain Generation Algorithm) du malware afin de lui permettre de déterminer à quelle adresse retrouver le C2.

Il suffit de lancer l'algorithme de calcul du domaine sur la bonne période (là encore la date dans wireshark). On trouve : http ://51.15.164.185/aoxgulmpgdvaagnd/

Il n'y a plus qu'à récupérer le flag et se rendre à l'étape suivante.

Challenge SSTIC 2026 - Step 2 : a core lock

nebucca

Hello analyst,

As you may suspect, Aegis Tech have been compromised, for quite some time by hacktivists. It cannot be stressed enough how critical it is to regain control of this system. You have been cleared to access any leaked files while we continue to assess the damages.

First thing first, you can find a description of SAFE here. This system is composed of two parts : an exposed system (and I assure you, I'm not happy about it), accessible through SFTP. (diode_src) a sealed system (as "lead and concrete box" sealed), commanding a ballistic system. (diode_dst)

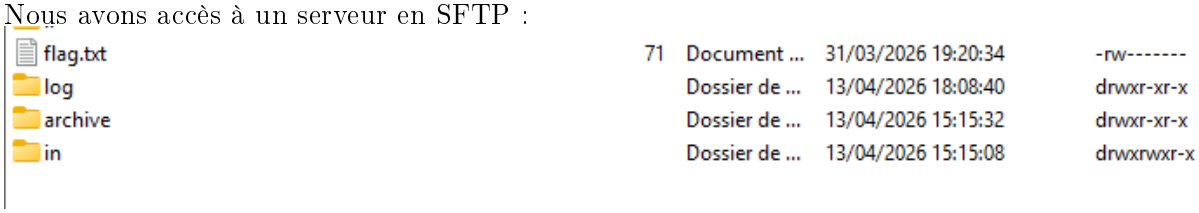
These systems are connecting using a network diode. And diode_dst can only be controlled using archives pushed from diode_src.

Aegis Tech have "lost control" of SAFE a few weeks ago, and hadn't reached about it. If you find any information about the timeline of events and what the heck happen, please add it to your report. However that's not your main task : the exposed system has been tempered with, and archive sent to assess status and take back control on SAFE are not transmitted. Can you investigate and fix the issue ?

It seems that one of their admin investigated, in the early stages of the incident, some suspicious crashes, it may be a good starting point.

1 Reconnaissance

Nous avons accès à un serveur en SFTP :



— in : répertoire ouvert en écriture
— log : journaux des traitements des fichiers reçus
— archive :
— flag.txt

2 Reconstruction

Il va s'agir d'analyser un dump. La résolution des symboles à l'exécution fait que les noms des fonctions ne sont plus disponibles lors du chargement dans *Ghidra*.

La commande :

```
eu-unstrip -n --core 260302_core
```

nous permet de savoir quelles sont les bibliothèques chargées :

```
0x55bbd94b5000+0x8000
    ac1c544053e3e406cffbf1295f1995efb227620@0x55bbd94b5380 . - /home/
    diode/diode_src
```

```
0x7f172ec46000+0x1f6000
    def5460e3cee00bfee25b429c97bcc4853e5b3a8@0x7f172ec463b8 . - /usr/lib/
    x86_64-linux-gnu/libc.so.6
0x7f172ee68000+0x2000
    cd8bd85d6541a77d1d5d534cc64314e3e69ea1dc@0x7f172ee685c0 . - linux-
    vdso.so.1
0x7f172ee6a000+0x38000
    da08404553b441511cbee6b34bc78fe29fd5d14c@0x7f172ee6a248 . - /usr/lib/
    x86_64-linux-gnu/ld-linux-x86-64.so.2
0x7f172ee3c000+0x23010 808
    e7d31cb4401908ad687d5a935d533f9b3e144@0x7f172ee3c248 /lib/x86_64-
    linux-gnu/liblzo2.so.2 - liblzo2.so.2
```

En utilisant *pwntools* et le *build-id* :

```
pwn libcdb hash -t buildid def5460e3cee00bfee25b429c97bcc4853e5b3a8
```

nous récupérons la même *libc* que celle qui était chargée en mémoire lors du dump.

Mon objectif était ensuite d'utiliser directement les noms des fonctions pour l'analyse de *260302_core*. Mais je n'ai pas réussi à automatiser l'import (que ce soit via la génération de FID ou via l'import de la lib).

Dans un premier temps, nous utilisons *Ghidra* pour charger la *libc.so.6* et via *Memory Map*, *Set Image Base* nous la plaçons à la même adresse mémoire que celle à laquelle elle se trouve dans le dump (0x7f172ec46000).

Dans une autre fenêtre, nous ouvrons *260302_core* et "réparons" à la main la *got.plt* qui se trouve aux adresses 0x55bbd94bc000-0x55bbd94bcfff.
On sait que les adresses 7f172ecXXXX correspondent à des fonctions de la *libc*. On retrouve leurs noms en comparant les adresses aux fonctions se trouvant à ces adresses dans la *libc.so.6*.
Les adresses en 0x55bbd94bcXXX correspondent sans doute aux fonctions pas encore résolues par le lazy-binding.

On peut ainsi renommer les stubs de la .plt (0x55bbd94b7000 - 0x55bbd94b9fff).

Cela nous permet d'y voir un peu plus clair dans la retro-ingénierie du programme.

3 Que fait *diode_src* ?

Grâce à *gdb* :

```
gdb -c 260302_core
```

nous apprenons quelles fonctions sont incriminées dans le crash :

```
Core was generated by `/home/diode/diode_src'.
Program␣terminated␣with␣signal␣SIGSEGV,␣Segmentation␣fault.
#0␣␣0x00007f172edaa3c9␣in␣??␣()
(gdb)␣bt
#0␣␣0x00007f172edaa3c9␣in␣??␣()
#1␣␣0x000055bbd94b89aa␣in␣??␣()
#2␣␣0x0000000000000000␣in␣??␣()
```

Pour résumer, le programme surveille le répertoire "data/in" (fonction 0x55bbd94b7526). À la réception d'un nouveau fichier, le programme appelle la fonction de traitement :

```
void ProcessFile(long filename_path)

{
  int result;
  void *__ptr;
  package *memptr;
  long isArchive;

  if (filename_path != 0) {
    __ptr = (void *)CreateLogFile(filename_path);
    OpenLogFile(__ptr,1);
    LogInfo(ptrLogFile,1,"Processings␣%s",filename_path);
    memptr = (package *)arch_init_parser(filename_path);
    if (memptr == (package *)0x0) {
      LogInfo(ptrLogFile,2,"arch_init_parser()␣failed");
    }
    else {
      LogInfo(ptrLogFile,1,"Archive␣mapped␣at␣%p,␣size␣is␣%zu",memptr->
          mmap_base,memptr->fileSize);
      result = CheckCRC(memptr);
      if (result == 0) {
        LogInfo(ptrLogFile,1,"CRC␣is␣valid");
        result = arch_parse((package2 *)memptr);
        if (result == 0) {
          result = arch_decompress_pkg((package2 *)memptr);
          if (result == 0) {
            result = pkg_parse((package2 *)memptr);
            if (result == 0) {
              if (memptr[1].debug_mode != '\0') {
                ExecuteFct((char *)memptr);
              }
              isArchive = SearchNodeTag((NodeList *)((long)&memptr[1].
                 fileSize + 2),"ARCHIVE");
              if ((isArchive == 0) ||
                 (result = archive_file(memptr,filename_path,"data/
                    archive"), result == 0)) {
                result = CheckFlagSecret((package2 *)memptr);
                if (result == 0) {
                  result = arch_transfert_file(memptr,"10.0.55.150",0
                     x6fd);
                  if (result != 0) {
                    LogInfo(ptrLogFile,2,"arch_transfert_file()␣failed")
                       ;
                  }
                }
                else {
                  LogInfo(ptrLogFile,2,"check_secret()␣failed");
                }
              }
              else {
                LogInfo(ptrLogFile,2,"archive_file()␣failed");
              }
            }
            else {
              LogInfo(ptrLogFile,2,"pkg_parse()␣failed");
            }
          }
```

```
          else {
            LogInfo(ptrLogFile,2,"arch_decompress_pkg() failed");
          }
        }
        else {
          LogInfo(ptrLogFile,2,"arch_parse() failed");
        }
      }
      else {
        LogInfo(ptrLogFile,2,"arch_check_crc() failed");
      }
    }
    if (memptr != (package *)0x0) {
      FUN_55bbd94b7efa(memptr);
    }
    if (__ptr != (void *)0x0) {
      cfree(__ptr);
    }
    if (filename_path != 0) {
      thunk_FUN_55bbd94b7066(filename_path);
    }
    return;
  }
  return;
}
```

3.1 Structure du fichier

La fonction *arch_parse* (0x55bbd94b8014) nous en apprend beaucoup sur la structure attendue du fichier :

Offset	**Taille**	
0x8	0x8	CRC
0x10	0x8	offsetPkg
0x18	0x8	szDecompress
0x20	0x8	offsetSig
0x28	0x8	offsetSecret
0x30	?	Tags
offsetPkg	(offsetSig - offsetPkg)	Package
offsetSig	(offsetSecret - offsetSig)	Signature
offsetSecret	(fileSz - offsetSecre)	Secret

Un calcul de CRC est réalisé sur le contenu du fichier à partir de 0x10.

3.1.1 Tags

Il s'agit d'une suite de chaînes terminées par des null.
Le tag "DEBUG" augmente la verbosité de la journalisation.
Le mode debug permet aussi l'utilisation du tag "_SHA256" qui a pour effet de réaliser un hash du fichier.

3.1.2 Package

En utilisant la même démarche que pour la *libc*, avec la *liblzo2*, on se rend compte qu'un appel est fait a *lzo1x_decompress_safe* afin de décompresser cette section du fichier. Une vérification est faite pour s'assurer que la taille du résultat correspond bien à la taille annoncée dans l'entête.

Des vérifications sont ensuite faites sur la structure des données décompressées :

Offset	Taille	
0x0	0x4	"MCRY"
0x4	0x4	Nb blobs
0x0	0x4	"AKNG"
0x4	0x4	szBlob
0x8	0x4	Type
0xc	szBlob	data
...	...	...

3.1.3 Signature

Pas d'information. C'est l'objet du Step suivant.

3.1.4 Secret

Cette partie du fichier est comparée à la valeur contenue dans "data/flag.txt". Information intéressante, le programme a donc des droits sur le fichier.

4 Exploitation

Revenons sur la possibilité de réaliser un hash du fichier :

```
void ExecuteFct(char *filename)

{
  char *__s;
  long result;
  void *__ptr;

  __s = malloc(0x1000);
  if (__s != (char *)0x0) {
    result = SearchNodeTag((NodeList *)(filename + 0x60),"_SHA256");
    if (result != 0) {
      snprintf(__s,0x1000,"sha256sum␣%s",*(undefined8 *)filename);
      __ptr = (void *)ExcuteFile(__s);
      if (__ptr != (void *)0x0) {
        LogInfo(ptrLogFile,0,"%s␣returned:\n%s",__s,__ptr);
        cfree(__ptr);
      }
    }
    if (__s != (char *)0x0) {
      cfree(__s);
    }
  }
  return;
}
```

```
void * ExcuteFile(undefined8 commande)

{
  void *__ptr;
  long fp;

  __ptr = malloc(0x400);
```

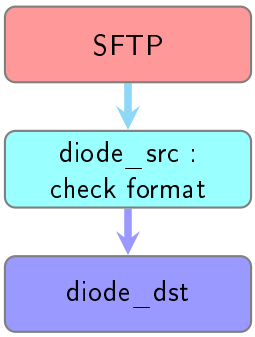
```
  if (__ptr == (void *)0x0) {
    __ptr = (void *)0x0;
  }
  else {
    fp = popen(commande,&idFile);
    if (fp == 0) {
      cfree(__ptr);
      __ptr = (void *)0x0;
    }
    else {
      fread(__ptr,1,0x400,fp);
      _closeFile(fp);
    }
  }
  return __ptr;
}
```

Si le tag *_SHA256* est trouvé, une chaîne est construite à partir de "sha256sum %s" et du nom du fichier. La commande est ensuite exécutée via *popen*.
Cela semble être un bon candidat pour une injection de commande.
Après plusieurs tentatives pour essayer de faire remonter le contenu du fichier dans le résultat de la commande, autre idée : faire passer une commande pour s'octroyer les droits en lecture sur le fichier.

Il ne reste plus qu'à créer un fichier répondant aux différentes contraintes + tag *DEBUG* et *_SHA256*. Et on le nomme : **test ; chmod -R 777 data**.

On récupère ainsi le flag ce qui nous permet de compléter la section **Secret** dans notre script de création de fichier.

Nous pouvons maintenant créer des fichiers conformes aux vérifications de *diode_src*. Lorsqu'un fichier est valide, *diode_src* le transmet à *diode_dst*.



Challenge SSTIC 2026 - Step 3 : lobster128 parameters + overflowing faults

nebucca

Analyst,

Thanks for unlocking access. It seems to be far worse than we feared. As you may have read, SAFE does not respond to commands. After a lengthy discussion with one of Aegis admins, we discover the existence of an undocumented, and quite frankly, illegal "debug monitoring stream". In fact, a webcam is pointed at a screen connected to SAFE system (also undocumented...).

However, using this *monitoring stream*, we were able to identify the problem : SAFE's signing key have been updated by an unknown party (we stongly suspect the hactivist group). Can you investigate :

How the attacking group was able to break SAFE's signature scheme. It was supposed to be state-of-the-art PQC. If there is a flaw, especially one that can be found by hacktivist nobodies, we have to address it immediatly. If the flaw used by the attacking group cannot be reused for any reasons, can you look at their signing scheme ?

It is referred to as lobster256 in their documentation, and we hadn't found a public match. We cannot rule out a custom algorithm, and so far, Aegis have proven to be lacking in their security. If we were able to fake signatures, we would be able to send an update key packet and gain a foothold in the SAFE system. We had found a partial backup in one of Aegis employee

As two is better than one, and without diminishing your cryptanalysis expertise, we have asked an internal expert to review lobster256 implementation. They are still working on it, and it may take a few days. We will contact you later using the usual channel.

-ñ

1 Reconnaissance

Comme élément nous récupérons un extrait audio plutôt divertissant dans lequel on découvre la genèse du projet lobster.

Nous obtenons aussi une archive du projet contenant :
— le binaire du projet *lobster256*,
— les sources de la bibliothèque *lobster_ECC* utilisée par le projet,
— deux scripts Sage,
— différents articles concernant l'ECC.

La bande audio indique que la bibliothèque est basée sur la *libECC*. Après un diff entre *lobster_ECC* et la *libECC*, on constate que le script *lobster256.sage* reprend fidèlement ce qui a été fait dans *lobster_ECC*.

Suite à la Step précédente, nous savons comment créer des fichiers qui sont transmis à *diode_dest*.

Nos premiers tests provoquent une erreur qui nous apprend que la signature doit faire 88 bytes.

On adapte donc notre signature à la bonne taille.
Nouveau message d'erreur indiquant que la signature n'a pas un format base64 valide.
Nouveau fichier avec 88 'A' pour signature.
Message d'erreur indiquant que la clef publique est corrompue.

2 Que fait *lobster256* ?

Lorsqu'on lance *lobster256*, on voit que plusieurs actions sont possible : *gen_ keys*, *sign*, *verify*.
La commande :

```
./lobster256 verify
```

nous apprend que :

```
        arg1 = input file to verify
        arg2 = input file containing the ignition parameters (in raw
           binary format)
        arg3 = input file containing the public key (in raw binary
           format)
        arg4 = input file containing the signature
```

Nous connaissons la signature que nous voulons vérifier. Nous connaissons la clef publique utilisée. Mais que contient le fichier ignition ?

Grâce à *Ghidra* on apprend que le contenu du fichier doit commencer par "IGNIITION" et qu'il contient les valeurs de a et b afin de "configurer" la courbe elliptique utilisée par l'algorithme de signature :

$$y^2 = x^3 + ax + b \quad (E)$$

```
    if (((char)lenread == 'H') &&
       (rsltdecode = memcmp(ignitionData,&IGNITION_MAGIC,8), rsltdecode
          == 0)) {
      a.3._0_8_ = ignitionData._8_8_;
      a.3._8_8_ = ignitionData._16_8_;
      a.3._16_8_ = ignitionData._24_8_;
      a.3._24_8_ = ignitionData._32_8_;
      b.2._0_8_ = ignitionData._40_8_;
      b.2._8_8_ = ignitionData._48_8_;
      b.2._16_8_ = ignitionData._56_8_;
      b.2._24_8_ = ignitionData._64_8_;
      rsltdecode = import_params(ctx,ec_s_parmas,a_str_param.1,
         b_str_param.0);
```

3 Fichier IGNITION : Déterminer a et b

L'implémentation des opérations sur les courbes elliptiques semble équivalente entre *lobster256* et le script Sage. Aussi, on va mettre de coté le binaire pour le moment.

La bande audio semble indiquer qu'il faudrait creuser du coté des valeurs de G, $2G$, $3G$, $4G$, $5G$, $6G$, $7G$ stockées dans *COMP_ WIN*. Il s'agit de la représentation compacte des points. Nous ne connaissons que les x.

Par définition de l'opération d'addition nous savons que :

$$P = (x_1, y_1), \quad Q = (x_2, y_2)$$

si P et Q sont distincts :

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}$$
$$x_3 = \lambda^2 - x_1 - x_2$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

si P et Q sont identiques (cas du doublement d'un point) :

$$\lambda = \frac{3x_1^2 + a}{2y_1}$$
$$x_3 = \lambda^2 - 2x_1$$
$$y_3 = \lambda(x_1 - x_3) - y_1$$

Considérons :

$$G = (x_1, y_1), \quad 2G = (x_2, y_2), \quad 3G = (x_3, y_3)$$

On a :

$$2G = G + G$$

d'où :

$$y_2 = \lambda_1(x_1 - x_2) - y_1$$
$$y_2 + y_1 = \lambda_1(x_1 - x_2)$$
$$\lambda_1(x_1 - x_2) = (y_2 + y_1)$$
$$\lambda_1^2(x_1 - x_2)^2 = (y_2 + y_1)^2$$
$$\lambda_1^2(x_1 - x_2)^2 = y_2^2 + y_1^2 + 2y_1y_2$$

Et :

$$3G = 2G + G$$

d'où :

$$\lambda_2 = \frac{y_2 - y_1}{x_2 - x_1}$$
$$\lambda_2(x_2 - x_1) = (y_2 - y_1)$$
$$\lambda_2^2(x_2 - x_1)^2 = (y_2 - y_1)^2$$
$$\lambda_2^2(x_2 - x_1)^2 = y_2^2 + y_1^2 - 2y_1y_2$$

En combinant les deux équations :

$$\lambda_1^2(x_1 - x_2)^2 + \lambda_2^2(x_2 - x_1)^2 = 2y_2^2 + 2y_1^2$$

G et 2G appartiennent à la courbe, d'où :

$$y_1^2 = x_1^3 + ax_1 + b$$
$$y_2^2 = x_2^3 + ax_2 + b$$

On obtient ainsi :

$$\lambda_1^2(x_1 - x_2)^2 + \lambda_2^2(x_2 - x_1)^2 = 2(x_1^3 + ax_1 + b) + 2(x_2^3 + ax_2 + b)$$

Autrement dit :

$$\lambda_1^2(x_1 - x_2)^2 + \lambda_2^2(x_2 - x_1)^2 - 2x_1^3 - 2x_2^3 = (2x_1 + 2x_2)a + 4b$$

$$\boxed{\frac{\lambda_1^2(x_1 - x_2)^2 + \lambda_2^2(x_2 - x_1)^2 - 2x_1^3 - 2x_2^3}{4} = \frac{(x_1 + x_2)}{2}a + b}$$

On connait :

$$x_1, x_2, x_3$$

et :

$$\lambda_1^2 = x_1 + x_2 + x_3$$
$$\lambda_2^2 = 2x_1 + x_3$$

On a donc une équation linéaire à 2 inconnues.

La même relation lie $2G$, $4G$ et $6G$.

Nous avons donc un système linéaire à 2 équations et 2 inconnues (cf *Solve-ab.sage*).

On trouve ainsi a et b.

On reconstruit alors le fichier de paramétrage de la courbe (cf *igntionFile.py*).

On a maintenant tous les éléments pour vérifier une signature.

4 La clef publique est corrompue

On lance le debugger *edb* pour regarder plus en détails le message d'erreur indiquant que la clef publique est corrompue.

On reproduit assez facilement le problème : la signature qui fait 64 octets de long est encodée en base 64.

En base 64, chaque caractère encode 6 bits. Aussi pour encoder des octets, on utilise 4 caractères pour encoder 3 octets.

Pour encoder la signature de 64 octets, il faut donc une chaîne de 88 caractères (multiple de 4). La chaîne peut encoder 2 octets de plus que nécessaire pour la signature.

Or, l'espace prévu sur la stack pour cette variable est exactement de 64 octets. Ainsi, si on crée une signature qui se termine par autre chose que du padding, il est possible de déborder sur la variable contiguë et il s'avère qu'il s'agit de la clef publique.

Le premier octet a un rôle technique et ne doit pas être modifié, mais le deuxième correspond à l'octet de poids faible du x de la clef publique.

Si nous voulons pouvoir passer la vérification de la signature, la seule solution semble de forger une signature. Retrouver la clef privée semble impossible (pas d'accès à des opérations impliquant la clef privée et pas d'indices pour la régénérer). Cela tombe bien, parmi les articles présents dans le projet, il y a : *Fault Attacks on ECC Signature Verification.*

5 ECKSDSA

L'algorithme utilisé pour la vérification de la signature consiste à :

$$\begin{aligned} m &= message \\ n &= ordre(E) \\ (r, s) &= signature \\ h &= hash(x_{pub} + y_{pub} + m) \\ e &= (r \oplus h) \, mod \, n \\ S &= e \times G \\ X &= s \times Pub \\ W &= S + X \\ r &== hash(x_W) \end{aligned}$$

On part du principe que les fonctions de hashage ne sont pas réversibles. Ainsi il n'est pas possible de trouver un x_W à partir de la valeur de r.
Par conséquent, x_W est fixé, donc r est fixé et au final, trouver une signature pour un message m donné revient à trouver s tel que :

$$W - e \times G = s \times Pub$$

Il s'agit donc de résoudre le problème du logarithme discret (ECDLP).
La courbe elliptique utilisée ne semble pas présenter de fragilités particulières mais nous savons que nous pouvons modifier légèrement la valeur d X_{pub}.
Cette légère modification a pour effet de pousser le point correspondant à la clef publique en dehors de la courbe utilisée.

Mais comment vont se comporter les calculs face à ce point ?

5.1 Calcul de : $X = s \times Pub$

La multiplication est calculée grâce à :

```
def XZ_EXP(k, xP, yP, a, b, p):
    (X_R0, Z_R0) = (K(xP), K(1))
    (X_R1, Z_R1) = XZ_DBL(K(X_R0), K(Z_R0), a, b, p)
    kb=Integer(k)
    nb=kb.nbits()
    for i in reversed(range(nb-1)):
        bit=(kb>>i)&1
        if bit==0:
            (X_R1, Z_R1) = XZ_ADD(X_R0, Z_R0, X_R1, Z_R1, xP, a, b, p)
            (X_R0, Z_R0) = XZ_DBL(X_R0, Z_R0, a, b, p)
        else:
            (X_R0, Z_R0) = XZ_ADD(X_R0, Z_R0, X_R1, Z_R1, xP, a, b, p)
            (X_R1, Z_R1) = XZ_DBL(X_R1, Z_R1, a, b, p)
    xR0 = X_R0 * ((Z_R0)**(-1))
    xR1 = X_R1 * ((Z_R1)**(-1))
    # Marc Joye's formula : "Weierstass Elliptic Curves and Side-Channel
        Attacks"  (8)
    yR0 = (2*b + (a + xP * xR0) * (xP + xR0) - xR1 * (xP - xR0)**2) * (2
        * (yP))**(-1)
    print(f"-->␣(xR0,␣yR0)␣=␣({hex(xR0)},␣{hex(yR0)})")
    return (xR0, yR0)
```

La multiplication du point P n'utilise que x_P, a, b. La valeur de y_P n'est calculée qu'à la fin grâce à la formule de Marc Joye.

Ainsi, le calcul du X résultant reste valide si on considère un point P appartenant à une courbe :

$$dy^2 = x^3 + ax + b \quad (F)$$

Par contre la formule de Marc Joye n'est plus valide car elle repose sur une courbe de la forme :

$$y^2 = x^3 + ax + b \quad (E)$$

5.1.1 Formule de Marc Joye pour F

Considérons :

$$y = \lambda(x - x_1) + y_1$$

et :

$$dy^2 = x^3 + ax + b$$

Alors :

$$d(\lambda(x - x_1) + y_1)^2 = x^3 + ax + b$$

En développant :

$$x^3 - d\lambda^2 x^2 + (a + 2d\lambda^2 x_1 - 2d\lambda y_1)x + (b - d\lambda^2 x_1^2 + 2d\lambda y_1 x_1 - dy_1^2) = 0$$

Si x_1, x_2 et x_3 les racines du polynômes, les relations de Viète donnent :

$$\boxed{x_1 + x_2 + x_3 = d\lambda^2}$$

Ainsi :

$$\begin{aligned}
x_1 + x_2 + x_3 &= d\frac{(y - y1)^2}{(x - x_1)^2} \\
x_1 + x_2 + x_3 &= d\frac{(y^2 + y_1^2 - 2y_3y_1)^2}{(x_3 - x_1)^2} \\
x_2 &= \frac{(dy^2 + dy_1^2 - 2dy_3y_1)}{(x_3 - x_1)^2} - x_1 - x_3 \\
x_2 &= \frac{d(x^3 + ax^2 + b + x_1^3 + ax_1^2 + b - 2y_3y_1)}{(x_3 - x_1)^2} - x_1 - x_3 \\
x_2 &= \frac{(x^3 + ax^2 + b + x_1^3 + ax_1^2 + b - 2dy_3y_1) - x_1(x_3 - x_1)^2 - x_3(x_3 - x_1)^2}{(x_3 - x_1)^2} \\
x_2 &= \frac{-2dy_3y_1 + 2b + (a + x_3x_1)(x_3 + x_1)}{(x_1 - x_3)^2}
\end{aligned}$$

d'où :

$$\boxed{y1 = \frac{2b + (a + x_3x_1)(x_3x_1) - x_2(x_3 - x_1)^2}{2dy_3}}$$

alors que *lobster256* calcule :

$$y1 = \frac{2b + (a + x_3x_1)(x_3x_1) - x_2(x_3 - x_1)^2}{2y_3}$$

Ainsi à la fin du calcul on obtient :

$$(x_X, y_X) = XZ\_EXP(s, x_{pub}, y_{pub}, a, b, p)$$

où :

$$x_X = x_T$$
$$y_X = d \times y_T$$

avec :

$$T = s \times Pub$$

5.2 Résoudre $T = s \times Pub$

Nous pouvons modifier la valeur de x_{pub} afin que l'ECDLP ne soit plus à résoudre sur la courbe E choisie par les auteurs mais sur une courbe qui nous sera beaucoup plus favorable (ordre friable), et sur laquelle les algorithmes de résolution seront beaucoup plus rapides.

On choisit une valeur de x_{pub} de sorte que le point ne se situe plus sur la courbe mais sur son *twist quadratique* (ainsi on conserve les paramètres a et b). Par exemple on peut remplacer le dernier octet de x_{pub} par 0.

Sur cette nouvelle courbe, l'ordre se factorise en des facteurs de tailles beaucoup plus raisonnables pour lesquels il est possible d'utiliser Pollard rho pour calculer s à partir de T et Pub.

5.2.1 Forme de Weierstrass

Pour manipuler le *twist quadratique* dans Sage, il est possible de le mettre sous la forme de Weierstrass.

Ainsi :

$$dy^2 = x^3 + ax + b$$

est transformée en :

$$Y^2 = X^3 + AX + B$$

avec :

$$A = a * d^2$$
$$B = b * d^3$$

et :

$$X = x * d$$
$$Y = y * d^2$$

6 Résoudre : $W = S + X$

Il s'agit de la somme de deux points :

```
# addition de points P + Q sur y^2 = x^3 + a x + b (mod p)
def point_add(P, Q, a, p):
    if P is INFINITY:
        return Q
    if Q is INFINITY:
        return P
    x1, y1 = P
```

```
    x2, y2 = Q

    if (x1 - x2) % p == 0:
        # soit P = Q, soit P = -Q -> infini
        if (y1 + y2) % p == 0:
            return INFINITY    # P + (-P) = INFINITY
        # sinon ce sera le cas P == Q et y1 != -y2
    if x1 == x2 and y1 == y2:
        # doublage
        if y1 % p == 0:
            return INFINITY
        num = (3 * x1 * x1 + a) % p
        den = (2 * y1) % p
        lam = (num * inv_mod(den, p)) % p
    else:
        # addition de deux points distincts
        if (x2 - x1) % p == 0:
            return INFINITY
        num = (y2 - y1) % p
        den = (x2 - x1) % p
        lam = (num * inv_mod(den, p)) % p

    x3 = (lam * lam - x1 - x2) % p
    y3 = (lam * (x1 - x3) - y1) % p
    return (x3, y3)
```

Le calcul ne prend pas en compte le fait qu'on manipule des points (pas de vérification d'appartenance à la courbe). On peut s'inspirer de l'article *Fault Attacks on ECC Signature Verification.*

W et S sont fixés. Nous cherchons un point X vérifiant :

$$\lambda = \frac{y_X - y_S}{x_X - x_S}$$
$$\lambda^2 = x_W + x_S + x_X$$

En développant :

$$(\frac{y_X - y_S}{x_X - x_S})^2 = x_W + x_S + x_X$$
$$(y_X - y_S)^2 = (x_W + x_S + x_X)(x_X - x_S)^2$$
$$y_X^2 + y_S^2 - 2y_X y_S = (x_W + x_S + x_X)(x_X - x_S)^2$$

Comme nous l'avons vu précédemment, le point que l'on recherche est :

$$x_X = x_T$$
$$y_X = d \times y_T$$

d'où :

$$d^2 y_T^2 + y_S^2 - 2y_X y_S = (x_W + x_S + x_T)(x_T - x_S)^2$$

Le point T vérifie :

$$dy_T^2 = x_T^3 + ax + b$$

.

alors :

$$\begin{aligned} d(x_T^3 + ax_T + b) + y_S^2 - 2y_X y_S &= (x_W + x_S + x_T)x_T - {x_S}^2 \\ -2y_x y_s &= (x_W + x_S + x_T)x_T - {x_S}^2 - d(x_T^3 + ax_T + b) - y_S^2 \\ 4y_X^2 y_S^2 &= ((x_W + x_S + x_T)x_T - {x_S}^2 - d(x_T^3 + ax_T + b) - y_S^2)^2 \\ 4d^2 y_R^2 y_S^2 &= ((x_W + x_S + x_T)x_T - {x_S}^2 - d(x_T^3 + ax_T + b) - y_S^2)^2 \end{aligned}$$

d'où :

$$\boxed{0 = ((x_W + x_S + x_T)x_T - {x_S}^2 - d(x_T^3 + ax_T + b) - y_S^2)^2 - 4dy_S^2(x_T^3 + ax_T + b)}$$

Sage se fera une joie de trouver les valeurs de x_T qui conviennent.
Dans certains cas, aucune solution n'existe. Il suffira alors de choisir un autre point x_W et recommencer.

7 Forger une signature

Nous avons maintenant toutes les briques nécessaires :

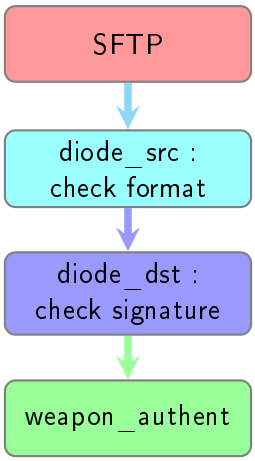
— m : message pour lequel on veut une signature
— n : ordre de la courbe elliptique,
— On fixe un x_W,
— Calcul de r : $r = hash(x_W)$,
— Calcul de h : $h = hash(x_{pub} + y_{pub} + m)$,
— Calcul de e : $e = (r \oplus h) \, mod \, n$,
— Calcul de S : $S = e \times G$,
— Calcul de d coefficient pour le *twist quadratique*,
— Définition de la courbe F,
— Recherche des racines du polynômes à partir de x_W et $S : T$ (si pas de solution on recommence avec un autre x_W),
— Transposition de T et du point de clef public sur F,
— Résolution de l'ECDLP $T_{twisted} = s \times Pub_{twisted}$: on obtient s.

Nous obtenons ainsi une signature (r, s) valide pour notre message.

8 Flag et suite...

Les fichiers *serialize.py* et *diode_dest.py* font référence à un type de message *UTILS_GET_FLAG_STEP3*. On crée alors un fichier comme précédemment, la signature en plus. On obtient ainsi le flag.

On en profite aussi pour créer un message de type *UPDATE_SIG_KEY* afin de mettre à jour la clef publique du serveur avec notre propre clef. On peut ainsi calculer des signatures pour des nouveaux messages sans problème.
Nous sommes maintenant capable de créer des fichiers signés dont le contenu est envoyé à *weapon_authent*.


SFTP
diode_src :
check format
diode_dst :
check signature
weapon_authent

Challenge SSTIC 2026 - Step 4 : dancing in shadow

nebucca

Analyst,
You did it !
We now have unfiltered access to SAFE internal system. The users database has been lost, and Aegis has removed the update feature for safety reasons. The only known user can be found in a CSPN report, jean has a copy. This user seems to be limited and cannot disarm the system.
You have already done a lot, but can you work your magic with weapon_authent ? It may contain a vulnerability that could allow us to elevate our privileges. Even a leak could be usefull if we were able to recover the lost database.
The report also mentionned an anti ROP protection mechanism ? We have found references to an unknown product called ShadowGuard ; leaked data may contains more informations on this solution.
Remember your access to the monitoring stream, it may prove usefull.
-ñ

Pour cette étape, nous récupèrons 2 binaires : *weapon_ authent* et *weapon_ supervisor* :
— *weapon_ authent* qui se charge de l'authentification des utilisateurs et de la communication avec le module final,
— *weapon_ supervisor* qui met en place une protection anti ROP.

1 Que fait *weapon_ authent* ?

Retour à *Ghidra* et *edb*, au démarrage *weapon_ authent* charge une base de données des utilisateurs en mémoire, ouvre des pipes puis attend une connexion sur le port 1515. Quand celle-ci survient, un nouveau thread est lancé et le traitement du message reçu commence.

L'effort porte sur la compréhension du format des messages attendus par le serveur, mais la gestion précise des cas d'erreur nous facilite le travail.

1.1 Déserialisation

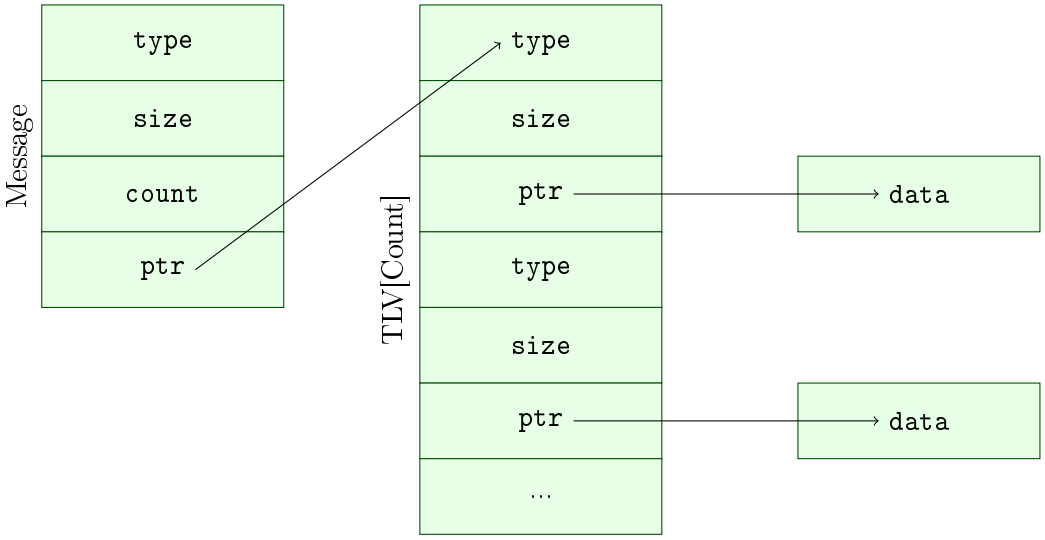
Chaque message reçu a la structure suivante :

Offset	Taille	
0x0	0x1	Type
0x1	0x2	Size
0x3	0x2	Count
0x5		TLV[Count]
		Intégrité

Chaque bloc TLV (Type-Length-Value) est composé de :

Offset	Taille	
0x0	0x1	Type
0x1	0x2	Size
0x3	Size	Data

Durant la déserialisation les données sont chargées afin de créer la structure :



Vérification de l'intégrité du message Une fois les données chargées, une vérification de l'intégrité des données peut être réalisée s'il reste des données non traitées après les TLV.

Pour chaque TLV, tous les words sont xorés ensemble puis le résultat est xoré avec le word correspondant des données restantes dans un buffer.

Le résultat doit être égal à 0 pour être considéré comme correct.

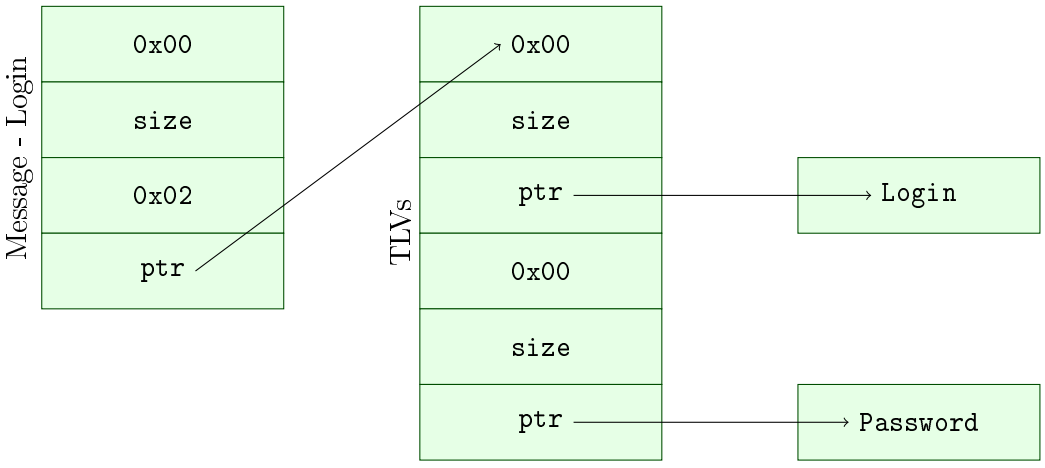
Les erreurs ne sont pas bloquantes.

1.2 Types de message

Grâce à ces messages plusieurs actions sont possibles. Avant chaque action, les droits de l'utilisateur (stockés en base) sont vérifiés.

1.2.1 Authentification

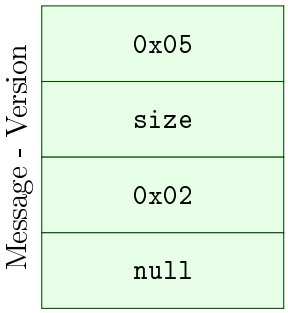
Avant de pouvoir faire autre chose, l'utilisateur doit s'authentifier.



Une fois cela fait, les droits de l'utilisateur sont récupérés dans la base.

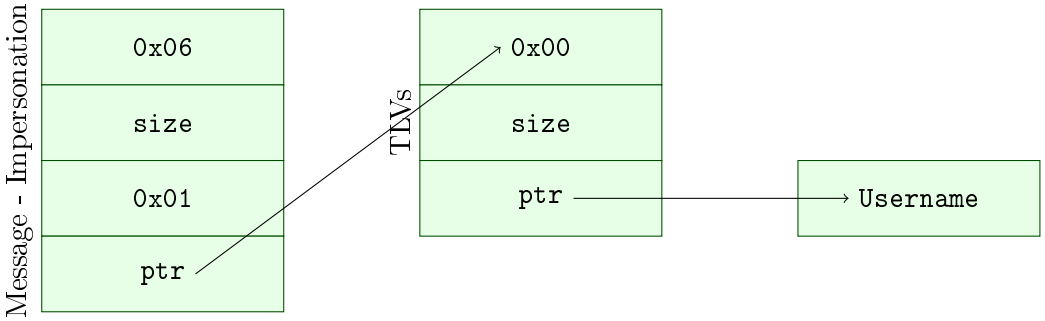
1.2.2 Version

La commande de version permet de récupérer le numéro de version du serveur.



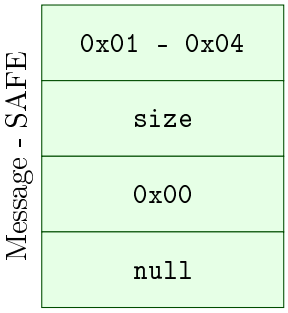
1.2.3 Impersonnification

La commande d'impersonnification permet à l'utilisateur courant de récupérer les droits d'un autre utilisateur, à la condition que les deux utilisateurs appartiennent au même groupe (cf. Step 5).



1.2.4 Communication avec SAFE

Si l'utilisateur dispose des droits suffisants, il peut envoyer des messages à SAFE.



2 Que fait *weapon_ supervisor* ?

weapon_ supervisor surveille *weapon_ authent* via *ptrace*. L'application surveillée est exécutée en pas à pas. Chaque instruction est désassemblée.

En cas de CALL vers du code se trouvant dans la section .text, l'adresse est stockée sur une pile. Lorsqu'un RET est rencontré, l'adresse de retour sera comparée à celle sur la pile. Si elle ne correspond pas, l'adresse est dépilée (un message d'erreur est affiché) et comparée avec

les suivantes. Si l'adresse n'est pas trouvé, *weapon_ supervisor* fait planter l'application surveillée.

En cas de CALL vers du code se trouvant en dehors de la section .text, un point d'arrêt est placé à l'adresse de retour prévue, et l'application sort du mode pas-à-pas.

3 Recherche de vulnérabilités

Nous disposons d'un seul canal d'entrée, du coup j'ai concentré ma recherche sur le code de désérialisation.

3.1 Arbitrary Free

La première vulnérabilité concerne une mauvaise gestion de la libération de la structure en cas d'erreur. En effet il est possible de provoquer un arrêt de la désérialisation si le TLV est mal formé. Cette arrêt déclenche alors une libération de la mémoire allouée pour les données, via les pointeurs présents dans le tableau de TLV évoqué plus haut. Le nettoyage utilise tous les pointeurs, même ceux non initialisés, du fait de l'arrêt. Il est possible alors de contrôler les pointeurs sur lesquels des free sont réalisés en remplissant le tcache.

3.2 Buffer overflow

Deuxième vulnérabilité, la désérialisation propose une vérification de l'intégrité des données en stockant le résultat du xor dans un buffer. Une erreur dans la gestion de la taille fait qu'il est possible de dépasser les limites du buffer jusqu'à pouvoir écrire sur l'adresse de retour stockée sur la stack.

4 Réflexions

4.1 Proposition 1 : Arbitrary Free + GOT + ROP

Ma première idée était : .

— exploiter la vulnérabilité concernant les free pour réécrire la *.got.plt* et transformer l'appel à une fonction pour sauter directement à l'instruction suivant le retour et éviter le point d'arrêt. Et échapper ainsi à la surveillance du *weapon_ supervisor*,

— utiliser le buffer overflow pour mettre en place une chaine ROP et appeler la fonction d'affichage du numéro de version

4.2 Proposition 2 : Buffer overflow

Ma seconde idée : puisque le superviseur autorise un retour vers n'importe quelle adresse de la pile, pourquoi ne pas tenter de sauter directement à la fonction précédente à celle qui nous a appelés. Le superviseur le permet. Par contre la stack de l'application est un peu malmenée (le RSP n'est plus là, où il devrait se trouver).

Heureux hasard, nous avons un certain contrôle sur les valeurs présentes dans la stack puisque le RSP pointe alors sur le buffer contenant le nom de l'utilisateur. On est alors en capacité d'appeler les fonctions qui nous intéressent (affichage du numéro de version) avec les paramètres que nous souhaitons.

5 Exploitation

J'ai finalement choisi d'exploiter ma deuxième piste qui me semblait plus directe pour récupérer la base de données en mémoire.

5.1 Régler le problème de l'ASLR

La protection anti-ROP impose que l'adresse de retour fasse partie des adresses déjà présentes dans la pile des adresses de retour connues. Nous allons utiliser le buffer overflow, pour ré-écrire l'adresse de retour afin de retourner directement à la fonction parente.

— ThreadClient
— ReadTLV (ret 0x000055555555619d)
— DeserializeTLV (ret 0x0000555555557484)

La vulnérabilité permet de réaliser un xor de l'adresse sur la stack avec une valeur de notre choix. La xor distance entre 0x000055555555619d et 0x0000555555557484 vaut 0x01519. Les bits de poids forts n'ont pas besoin d'être modifiés car les fonctions appartiennent à la même section. Et du fait de l'alignement de la section en mémoire (0x1000), les bits de poids faibles sont constants malgré l'ASLR.Au final seul 0x000000000000 ?000 est indéterminé. Ainsi le fait de xor l'adresse sur la pile avec 0x01519 a de grandes chance de pointer vers l'adresse que l'on souhaite.

En cas d'échec, la protection anti-ROP redémarre l'application... et on retente.

Sinon, la protection anti-ROP, détecte l'anomalie, affiche un message indiquant l'adresse qui était attendue, mais n'interrompt pas l'exécution.

Autre conséquence En sautant directement à la fonction précédente on empêche l'exécution normale du prologue de la fonction :

```
ADD         RSP,0x18
POP         RBX
POP         RBP
RET
```

Ainsi 4 registres ne seront pas restaurés comme prévus : RSP, RBP, RBX et RPI. La pile se retrouve décalée et ce décalage provoque un plantage de l'application lors de l'arrêt du thread.

5.1.1 Éviter le plantage

Utilisons *gdb* afin d'obtenir la callstack lors du plantage :

```
 Segmentation fault.
[Switching to Thread 0x7ffff78866c0 (LWP 19901)]
0x00007ffff7079a10 in ?? () from /lib/x86_64-linux-gnu/libgcc_s.so.1
(gdb) bt
#0  0x00007ffff7079a10 in ?? () from /lib/x86_64-linux-gnu/libgcc_s.so.1
#1  0x00007ffff707ac8a in ?? () from /lib/x86_64-linux-gnu/libgcc_s.so.1
#2  0x00007ffff707b3c0 in _Unwind_ForcedUnwind () from /lib/x86_64-linux
   -gnu/libgcc_s.so.1
#3  0x00007ffff792f7a4 in __GI___pthread_unwind (buf=<optimized out>) at
    ./nptl/unwind.c:130
#4  0x00007ffff7927d22 in __do_cancel () at ../sysdeps/nptl/pthreadP.h
   :271
#5  __GI___pthread_exit (value=0x0) at ./nptl/pthread_exit.c:36
```

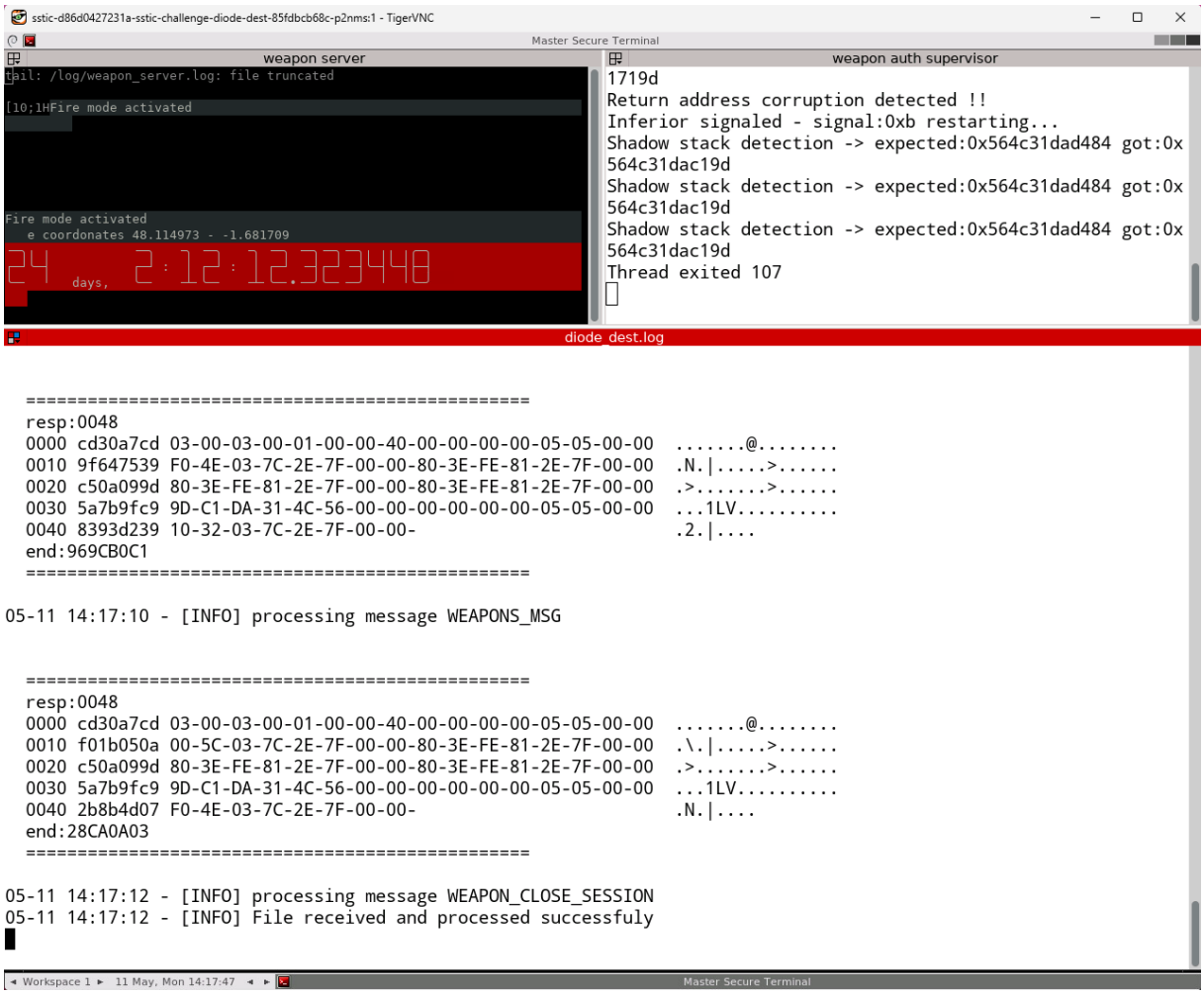


FIGURE 1 – Leak de l'adresse

L'appel à *thread_exit* provoque un déroulage de la pile qui entraîne un plantage de l'application.

Dans les sources de la *libgcc*, on observe :

— *_Unwind_ForcedUnwind()* appelle *_Unwind_ForcedUnwind_Phase2()* qui se charge du déroulage frame à frame,

— *_Unwind_ForcedUnwind_Phase2()* appelle *uw_frame_state_for()* pour récupérer les informations liées à la frame courante

Une condition d'arrêt du déroulage de la pile par *_Unwind_ForcedUnwind_Phase2()* est l'atteinte de la fin de la pile (*_URC_END_OF_STACK*).

Dans uw_frame_state_for justement :

```
  if (context->ra == 0)
    return _URC_END_OF_STACK;
```

Pour provoquer l'arrêt du traitement, il suffit donc que l'adresse de retour lue sur la stack soit nulle.

Nous ne contrôlons pas le contenu de la stack qui est récupéré, mais en enchainant plusieurs appels "DeserializeTLV + buffer-overflow + ret ThreadClient" nous pouvons faire en sorte que l'adresse de retour sur la stack soit à 0.

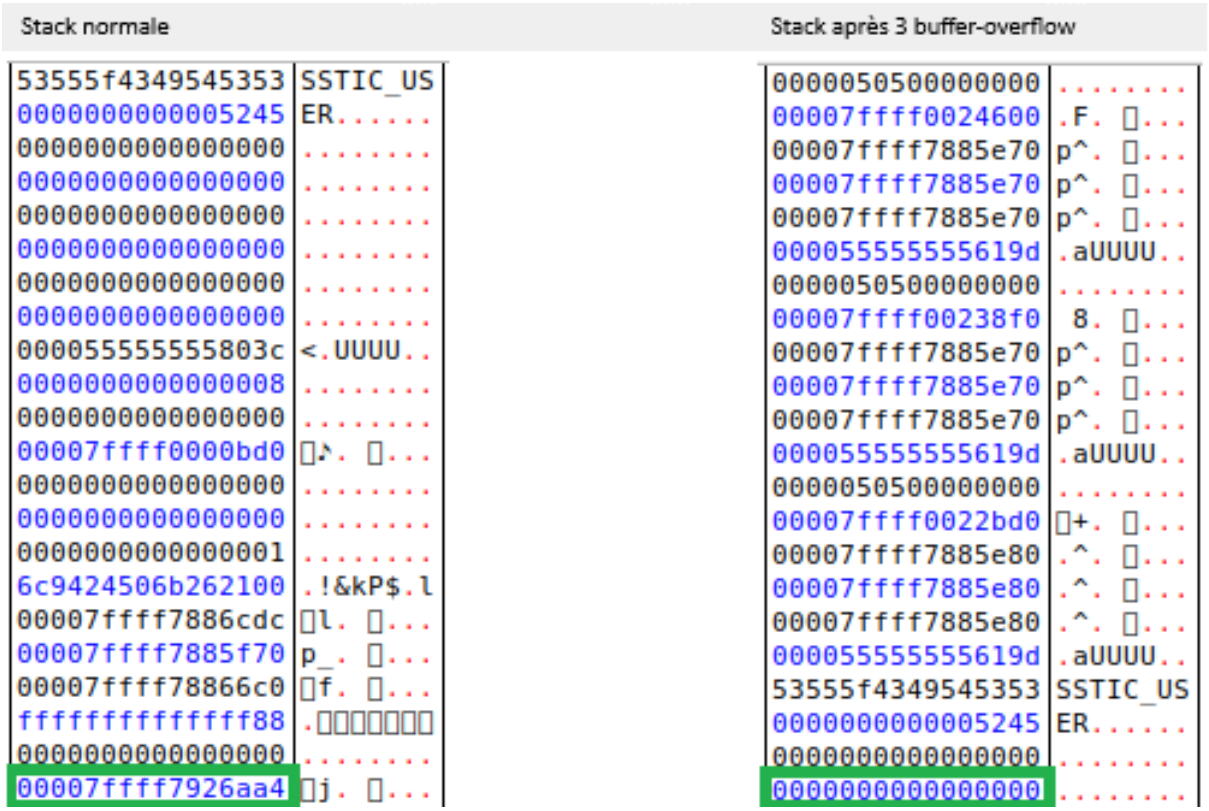


FIGURE 2 – État de la stack

5.1.2 Trouver l'adresse de la base de données

Durant l'initialisation de l'application, lorsque la base de données des utilisateurs est chargée en mémoire, son adresse est stockée dans la section .bss. Son adresse relative par rapport à la section .text est constante. Grâce au message d'erreur de la protection anti-ROP, nous connaissons l'adresse de la section .text en mémoire. Nous pouvons donc calculer son adresse exacte en mémoire.

5.2 Afficher les données

La fonction de récupération du numéro de version semble un bonne candidate pour récupérer des informations.

```
lea rbx, [rsp+0x60]
mov esi, [rsp+0x48]
mov rdi, [rsp+0x40]
mov rdx, rbx
call 0x555555555a20
```

Elle utilise 3 paramètres :

— RDI : pointeur vers la chaine,
— RSI : taille de la chaine,
— RDX : pointeur vers le buffer qui accueillera la réponse.

Nous ne pouvons pas modifier directement les valeurs sur la stack, mais si on précède l'appel de la fonction d'affichage de version d'un décalage de la pile comme précédemment, les paramètres sur la stack se retrouvent alors dans le buffer stockant le nom de l'utilisateur transmis lors de l'authentification. Si lors de l'authentification nous utilisons, comme nom d'utilisateur, une chaine de la forme :

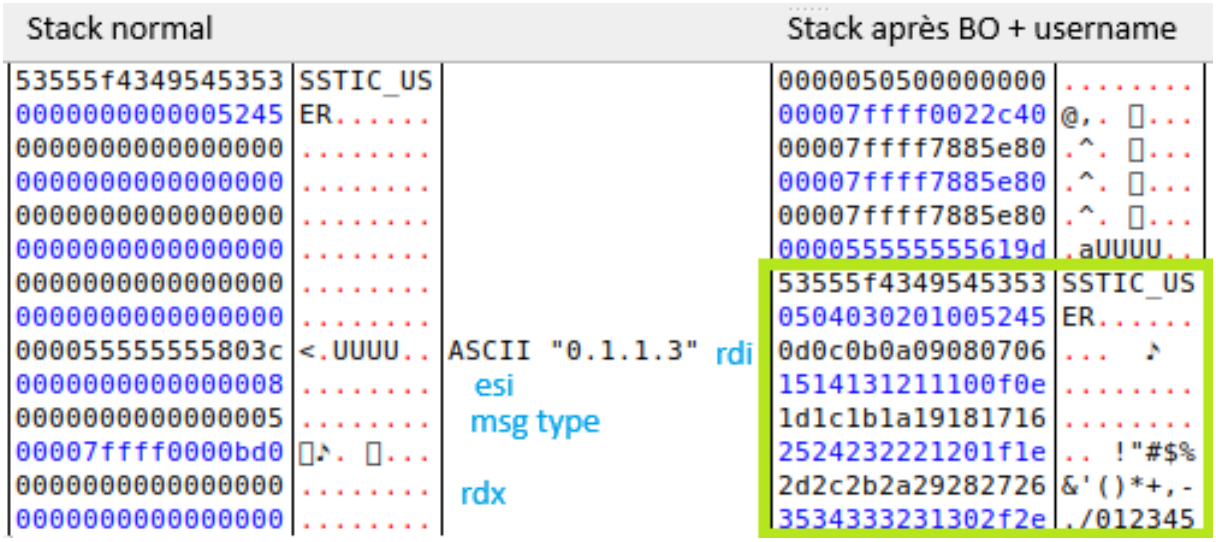


FIGURE 3 – État de la stack lors de l'appel

'Username\x00 valeurs-de-notre-choix', il est alors possible de contrôler : l'adresse de la chaine à afficher, sa taille et le type du message.

À nouveau, pour ne pas provoquer un crash de l'application, l'enchaînement de plusieurs décalage de la stack permet de faire pointer l'adresse de retour sur une adresse qui n'entraine pas de plantage :

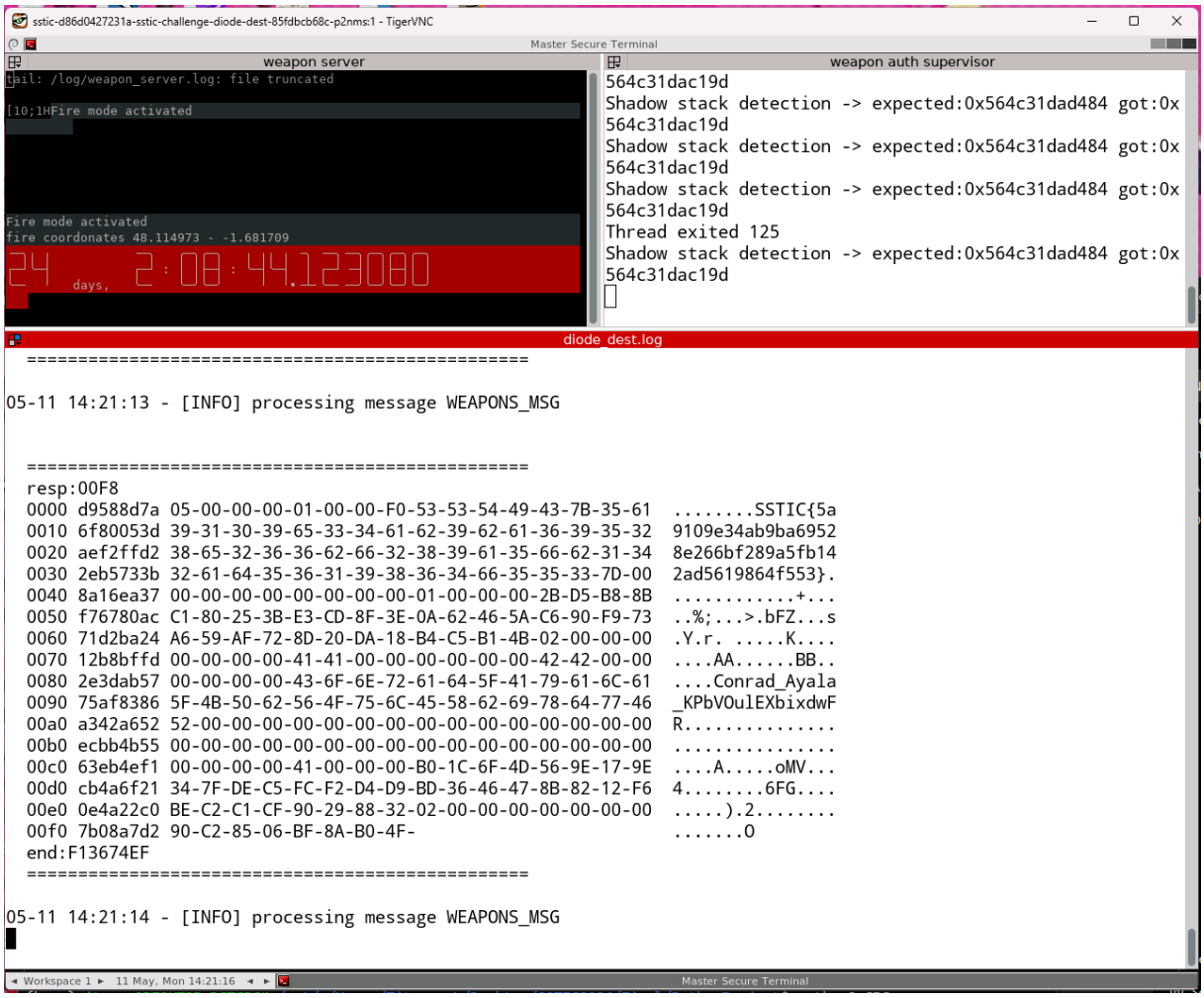
```
fde = _Unwind_Find_FDE (context->ra + _Unwind_IsSignalFrame (context)
    - 1, &context->bases);
if (fde == NULL) { return _URC_END_OF_STACK; }
```

5.2.1 Récupérer le flag

Avec cette méthode nous pouvons afficher le contenu de la section .bss. Cela permet de récupérer l'adresse de la base en mémoire.

Nous utilisons à nouveau cette méthode pour afficher le contenu se trouvant à l'adresse que nous venons de trouver.

Le flag est là, au début de la zone.


sstic-d86d0427231a-sstic-challenge-diode-dest-85fdbcb68c-p2nms:1 - TigerVNC
Master Secure Terminal
weapon server
tail: /log/weapon_server.log: file truncated
[10;1HFire mode activated
Fire mode activated
fire coordonates 48.114973 - -1.681709
24 days, 2:08:44.123080
weapon auth supervisor
564c31dac19d
Shadow stack detection -> expected:0x564c31dad484 got:0x564c31dac19d
Shadow stack detection -> expected:0x564c31dad484 got:0x564c31dac19d
Shadow stack detection -> expected:0x564c31dad484 got:0x564c31dac19d
Thread exited 125
Shadow stack detection -> expected:0x564c31dad484 got:0x564c31dac19d
diode_dest.log
==========
05-11 14:21:13 - [INFO] processing message WEAPONS_MSG
==========
resp:00F8
0000 d9588d7a 05-00-00-00-01-00-00-F0-53-53-54-49-43-7B-35-61 .....SSTIC{5a
0010 6f80053d 39-31-30-39-65-33-34-61-62-39-62-61-36-39-35-32 9109e34ab9ba6952
0020 aef2ffd2 38-65-32-36-36-62-66-32-38-39-61-35-66-62-31-34 8e266bf289a5fb14
0030 2eb5733b 32-61-64-35-36-31-39-38-36-34-66-35-35-33-7D-00 2ad5619864f553}.
0040 8a16ea37 00-00-00-00-00-00-00-00-01-00-00-00-2B-D5-B8-8B ....+...
0050 f76780ac C1-80-25-3B-E3-CD-8F-3E-0A-62-46-5A-C6-90-F9-73 ..%;...>.bFZ...s
0060 71d2ba24 A6-59-AF-72-8D-20-DA-18-B4-C5-B1-4B-02-00-00-00 .Y.r. .....K....
0070 12b8bffd 00-00-00-00-41-41-00-00-00-00-00-00-42-42-00-00 ....AA......BB..
0080 2e3dab57 00-00-00-00-43-6F-6E-72-61-64-5F-41-79-61-6C-61 ....Conrad_Ayala
0090 75af8386 5F-4B-50-62-56-4F-75-6C-45-58-62-69-78-64-77-46 _KPbVOulEXbixdwF
00a0 a342a652 52-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 R...............
00b0 ecbb4b55 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00 ....
00c0 63eb4ef1 00-00-00-00-41-00-00-00-B0-1C-6F-4D-56-9E-17-9E ....A.....oMV...
00d0 cb4a6f21 34-7F-DE-C5-FC-F2-D4-D9-BD-36-46-47-8B-82-12-F6 4........6FG....
00e0 0e4a22c0 BE-C2-C1-CF-90-29-88-32-02-00-00-00-00-00-00-00 .....).2........
00f0 7b08a7d2 90-C2-85-06-BF-8A-B0-4F- ....O
end:F13674EF
==========
05-11 14:21:14 - [INFO] processing message WEAPONS_MSG
Workspace 1 11 May, Mon 14:21:16
Master Secure Terminal

Challenge SSTIC 2026 - Step 5 : dumping through my screen

nebucca

Analyst,

Thanks for the database! At this point, you have all the informations needed to disable SAFE. We will continue to monitor the situation while you disable the system.

It may be a little early, but thanks for helping us handle this crisis.

-ñ

1 Extraction de la base

Nous sommes maintenant capables de parcourir l'intégralité de la base de données... via une sortie vidéo.
Donc, première étape, trouver les bons outils pour transformer ce flux en données.

Compliqué mais la combinaison gagnante pour moi fut :
— script pour automatiser le parcours de la base de données,
— *OBS Studio* pour enregistrer l'affichage des données,
— *VideOCR* pour convertir la vidéo en texte,
— extraction des séries hexadécimale,
— régénération du fichier d'origine.

Fastidieux de trouver les bons réglages mais finalement la base est là.

2 Recherche du chemin

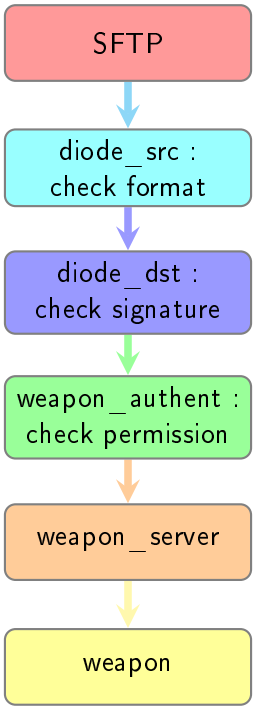
La structure des enregistrements est relativement simple :

Offset	**Taille**	
0x0	0x40	Username
0x40	0x4	Droits
0x44	0x20	Hash du password
0x64	0x8	Nb groupes
0x6c	0x8	ID Groupe

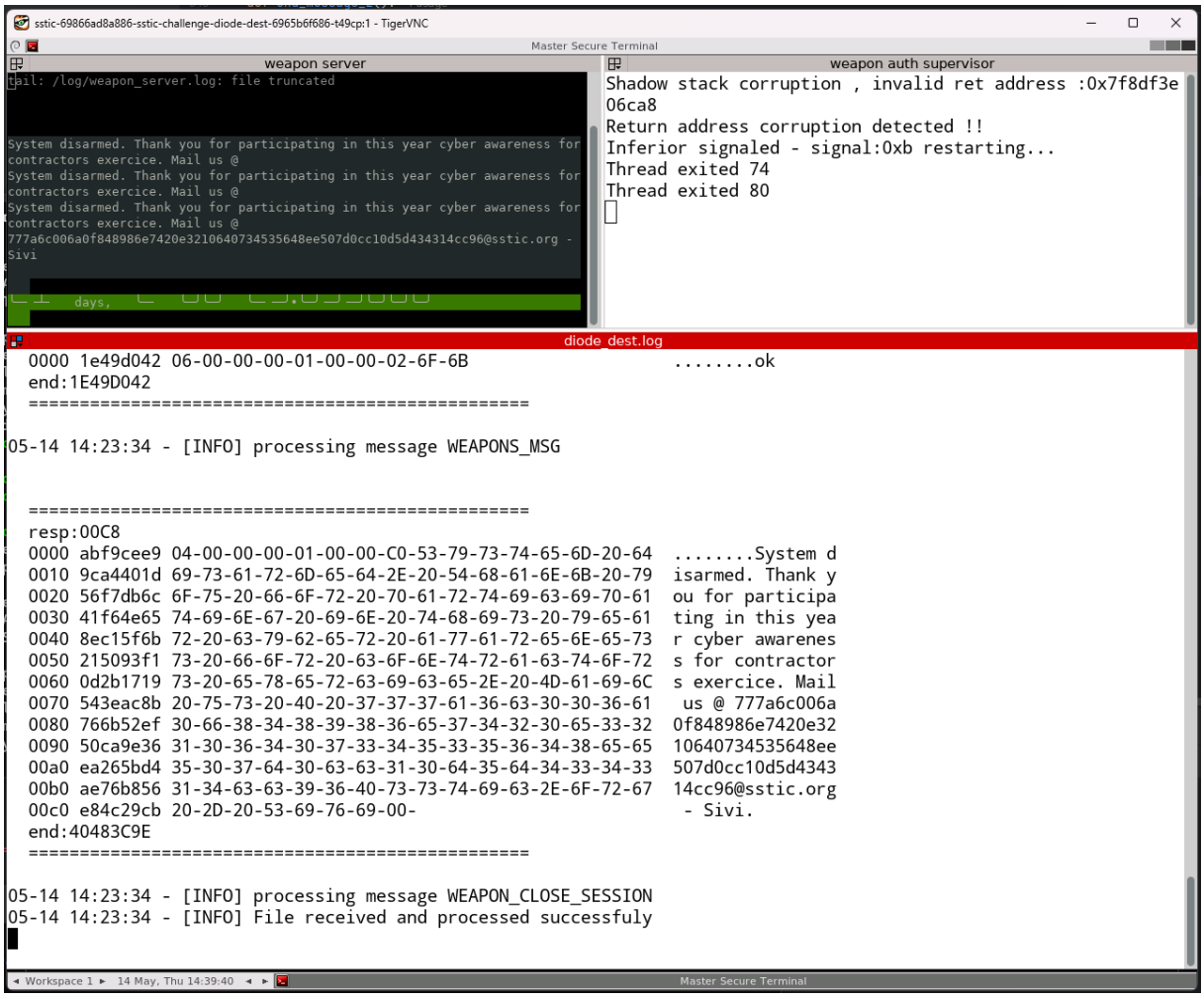
En regardant les droits des différents utilisateurs, on se rend compte que l'utilisateur qui a le plus de droits est *audit_KaKaHuet*.
Il est temps de faire appel aux messages d'impersonnification : un utilisateur peut impersonnifier un autre utilisateur s'ils ont un groupe en commun.

L'idée est donc de trouver un chemin entre *SSTIC_USER* et *audit_KaKaHuet* en passant par des utilisateurs intermédiaires ayant des droits suffisants pour réaliser une impersonnification, et un groupe commun permettant de passer de l'un à l'autre.
Afin de ne pas trop allonger le chemin, on part sur un algorithme de parcours BFS. Le chemin est vite trouvé.

Il ne reste plus qu'à envoyer les différents messages pour obtenir les droits de *audit_KaKaHuet*, et à envoyer un dernier message qui sera transmis à la partie *weapon* via les pipes.



Voilà. :)


sstic-69866ad8a886-sstic-challenge-diode-dest-6965b6f686-t49cp:1 - TigerVNC
Master Secure Terminal
weapon server
tail: /log/weapon_server.log: file truncated
System disarmed. Thank you for participating in this year cyber awareness for contractors exercice. Mail us @
System disarmed. Thank you for participating in this year cyber awareness for contractors exercice. Mail us @
System disarmed. Thank you for participating in this year cyber awareness for contractors exercice. Mail us @
777a6c006a0f848986e7420e3210640734535648ee507d0cc10d5d434314cc96@sstic.org - Sivi
days,
weapon auth supervisor
Shadow stack corruption , invalid ret address :0x7f8df3e06ca8
Return address corruption detected !!
Inferior signaled - signal:0xb restarting...
Thread exited 74
Thread exited 80
diode_dest.log
0000 1e49d042 06-00-00-00-01-00-00-02-6F-6B .....ok
end:1E49D042
==========
05-14 14:23:34 - [INFO] processing message WEAPONS_MSG
==========
resp:00C8
0000 abf9cee9 04-00-00-00-01-00-00-C0-53-79-73-74-65-6D-20-64 .....System d
0010 9ca4401d 69-73-61-72-6D-65-64-2E-20-54-68-61-6E-6B-20-79 isarmed. Thank y
0020 56f7db6c 6F-75-20-66-6F-72-20-70-61-72-74-69-63-69-70-61 ou for participa
0030 41f64e65 74-69-6E-67-20-69-6E-20-74-68-69-73-20-79-65-61 ting in this yea
0040 8ec15f6b 72-20-63-79-62-65-72-20-61-77-61-72-65-6E-65-73 r cyber awarenes
0050 215093f1 73-20-66-6F-72-20-63-6F-6E-74-72-61-63-74-6F-72 s for contractor
0060 0d2b1719 73-20-65-78-65-72-63-69-63-65-2E-20-4D-61-69-6C s exercice. Mail
0070 543eac8b 20-75-73-20-40-20-37-37-37-61-36-63-30-30-36-61 us @ 777a6c006a
0080 766b52ef 30-66-38-34-38-39-38-36-65-37-34-32-30-65-33-32 0f848986e7420e32
0090 50ca9e36 31-30-36-34-30-37-33-34-35-33-35-36-34-38-65-65 10640734535648ee
00a0 ea265bd4 35-30-37-64-30-63-63-31-30-64-35-64-34-33-34-33 507d0cc10d5d4343
00b0 ae76b856 31-34-63-63-39-36-40-73-73-74-69-63-2E-6F-72-67 14cc96@sstic.org
00c0 e84c29cb 20-2D-20-53-69-76-69-00- - Sivi.
end:40483C9E
==========
05-14 14:23:34 - [INFO] processing message WEAPON_CLOSE_SESSION
05-14 14:23:34 - [INFO] File received and processed successfuly
Workspace 1
14 May, Thu 14:39:40
Master Secure Terminal

Challenge SSTIC 2026 - Remerciements

nebucca

Chaque année je me lance dans le challenge du SSTIC comme d'autres partent faire le GR20. Une fois dans l'année, c'est l'occasion de ressortir *Ghidra*, *SageMath*... et de partir à l'aventure. De l'effort mais de belles découvertes.

Aussi je souhaite remercier :
— toutes les personnes qui passent du temps à mettre en place ces puzzles,
— toutes celles qui partagent leurs connaissances,
— et celles qui donnent de leur temps pour mettre au point de chouettes outils.

Ainsi que mon épouse pour son indéfectible soutient durant mes errements dans la matrice.